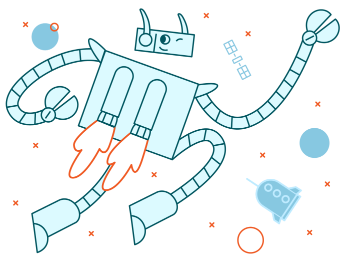
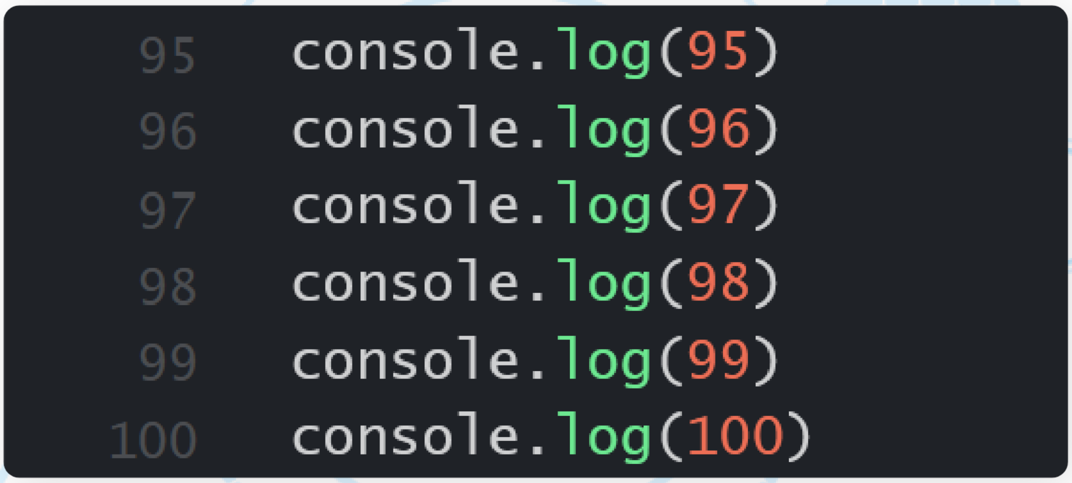
Loops



Loops are used to execute a block of code repeatedly without having to write that code multiple times.

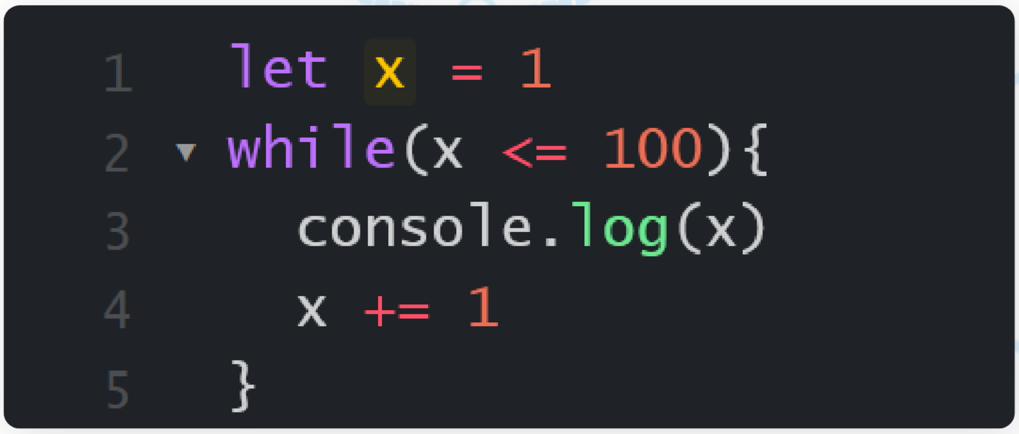
If we want to print the numbers 1 ~ 100 on the console, would you prefer to write 100 lines of `console.log`, or a loop statement to repeat your code?

100 lines of console.log:



```
console.log(95)
console.log(96)
console.log(97)
console.log(98)
console.log(99)
console.log(100)
```

A single `while` loop:



```
let x = 1
while(x <= 100){
  console.log(x)
  x += 1
}
```

While Loops

`while` loops will repeat the code while the condition is **True**.

Example:

```
let counter = 0
while (counter < 20) {
    counter++
    console.log("The current count is: " +
String(counter))
}
```

Using `while` loops for calculation:

```
let savings = 1000 // current bank savings: $1000
let workingDays = 0
while (savings < 2400) { // target savings of $2400
  workingDays += 1
  savings += 100
}

console.log("I have to work " + String(workingDays)
+ " days to save up $2400")
```

```
"I have to work 14 days to save up $2400"
```

For Loops

`for` loops will repeat the code for a set number of times.

A `for` loop statement is split into three parts

1. Initialise our variable with a number.
2. A comparison to check if condition is **True** or **False**.
3. If condition is **True**, increment our variable, run the block of code and go to the next iteration.

Example:

```
for (let counter = 0; counter < 10; counter++) {
    console.log("The current count is: " +
String(counter))
}
```

When to use While and when to use For

Generally, `while` loops are used when the number of times you need to iterate your code is **unknown**.

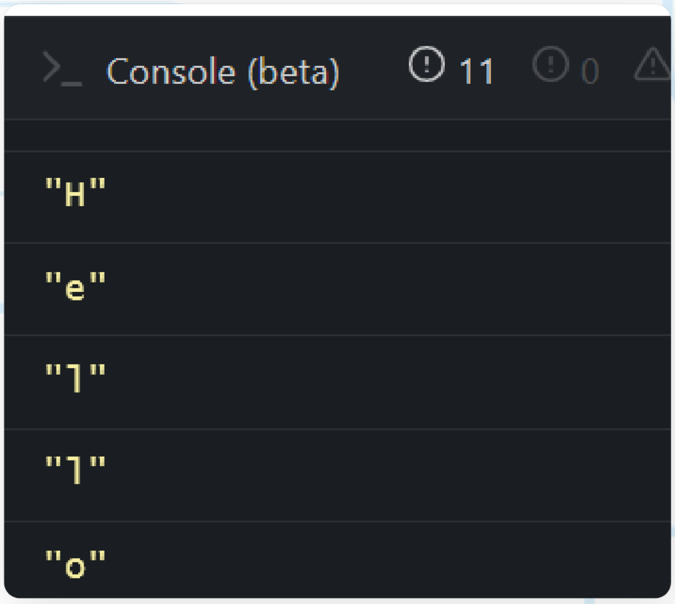
Use `for` loops when you **know** the number of iterations needed.

For...Of Loop

The `for...of` loop can be used to easily iterate each indexes or characters in a structure (Arrays, Strings, Maps).

Example:

```
let text = "Hello World"
for (let character of text) {
    console.log(character)
}
```



```
"H"
"e"
"l"
"l"
"o"
```

Break out of Loops

The `break` statement allows us to break out and end the loop early even if the condition for looping is still valid.

```
let counter = 0
while (counter < 5) {
  console.log(counter)
  if (counter == 3) {
    break
  }
  counter++
}
```

Output:

```
0
1
2
3
```

Skip a Loop

The `continue` statement will skip to the next iteration and disregard any code below.

```
let counter = 0
while (counter < 5) {
  counter++
  // if number is even, skip to next iteration
  if (counter % 2 == 0) {
    continue
    // any code after continue will be ignored
  }
  console.log(counter)
}
```

Output:

```
1
3
5
```

Nested Loops

Loops can be nested for solving and creating more complex algorithms. Like generating an English chessboard using 2-dimensional arrays.

Task: Print out the word 'Taco' in the following format

```
T
Ta
Tac
Taco
```

Solution: Nested for loops

```
let message = "Taco"
let length = message.length

for (let i = 0; i < length; i++) {
    let text = ""
  for (let j = 0; j <= i; j++) {
    text += message[j]
  }
  console.log(text)
}
```

Notice that this could also be solved with a single loop, using slices:

```
let message = "Taco"

for (let i = 0; i < message.length; i++) {
    console.log(message.slice(0, i + 1))
}
```

This goes to show that there are multiple ways to do things in JS :)